ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

(Laboratorio di)
Amministrazione di sistemi

LDAP

Marco Prandini

Dipartimento di Informatica – Scienza e Ingegneria

L'esigenza

- Un sistema correttamente funzionante dipende da parecchi file di configurazione, ma...
  - in una rete possono esserci centinaia di macchine che hanno gli stessi file di configurazione
  - aggiungere un utente, aggiungere un host, cambiare un parametro di funzionamento di un demone, vuol dire cambiare coerentemente centinaia di file di configurazione
- Allora ...
  - nasce l'esigenza di condividere queste informazioni tra sistemi diversi
  - l'approccio a file diventa inefficiente
- Occorre trovare un database centralizzato, in particolare
  - servizi di templating e configuration management
  - servizi centralizzati di autenticazione

Cosa non vedremo (purtroppo)

- Sistemi di descrizione delle infrastrutture cloud
  - rappresentano gli elementi costitutivi delle architetture
    - macchine virtuali
    - segmenti di rete
    - sistemi di gestione del traffico (firewall, bilanciatori, …)
    - servizi avanzati (db, storage a blocchi e a oggetti, …)
  - non come semplici dati di configurazione, ma come codice in grado di interagire con la piattaforma che li implementa
- Sistemi di configuration management
  - sono pensati per fornire modelli che descrivono lo stato di configurazione per un determinato ruolo (VM, servizio, …)
  - istanziano il modello sulla base delle specificità dell'obiettivo (es. hostname, ip, layout dei dischi, …)
- Sistemi di supporto a DevOps
  - integrano la gestione della configurazione infrastrutturale con gli strumenti di sviluppo e rilascio delle applicazioni
  - gestiscono l'individuazione dei servizi istanziati e l'instradamento delle richieste sulla base del loro stato

Gestione centralizzata dell'autenticazione

- Due aspetti da definire
  - come scegliere localmente la sorgente dei dati
  - come trasferire i dati da un server centrale ai sistemi client
- Scelta della sorgente
  - NSS: sistema generale per selezionare *name services*
  - PAM: sistema modulare per implementare *authentication modules*
- Protocollo di distribuzione
  - NIS/YP (obsoleto)
  - LDAP
- Tra i due: librerie con interfacce standard
  - libnss-ldap: permette di usare un db remoto LDAP come sorgente di nomi di qualsiasi tipo (utenti, gruppi, host, ...)
  - libpam-ldap: permette di accedere a un db remoto LDAP per autenticare utenti e ricavare regole di autorizzazione

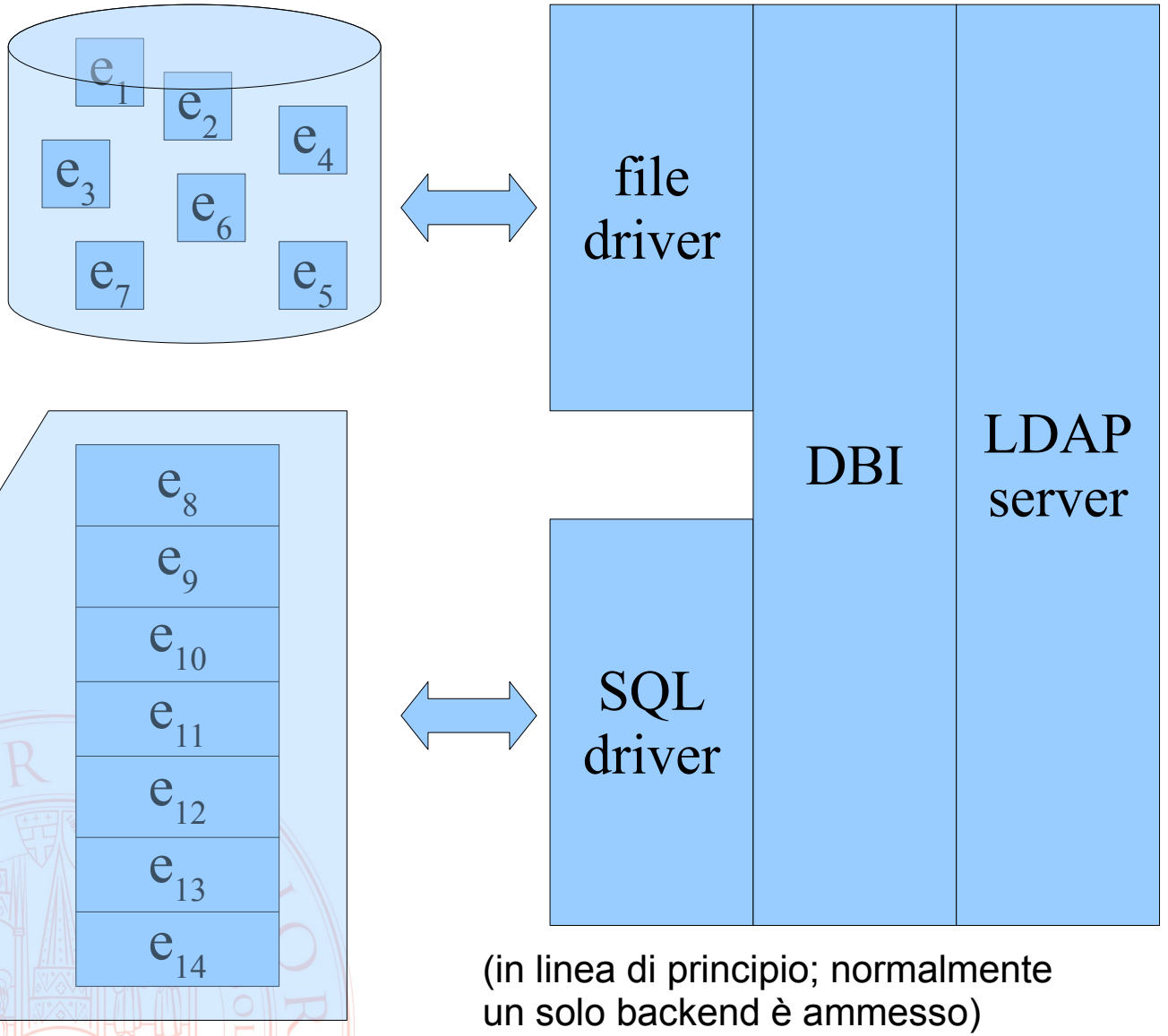
Perchè LDAP?

- implementa il concetto di directory service
  - standard ITU X.500
  - un database specializzato, sul modello "elenco del telefono"
    - gerarchico
    - #letture >> #scritture
    - non relazionale, scritture semplici → no transazioni
    - altamente disponibile → replica
- È uno standard maturo e manutenuto
  - Internet → tutti i sistemi **Unix-like**
    - RFC 1484/1485/1487/1488 *1993*
    - RFC 1777/1778/1779/1781 *1995*
    - RFC 2251..2256 *1997*
    - RFC 3377/3771 *2002*
    - RFC 4510..4519 *2006*
  - Dall'introduzione di Active Directory, è il sistema alla base delle **reti Microsoft** di tipo professionale

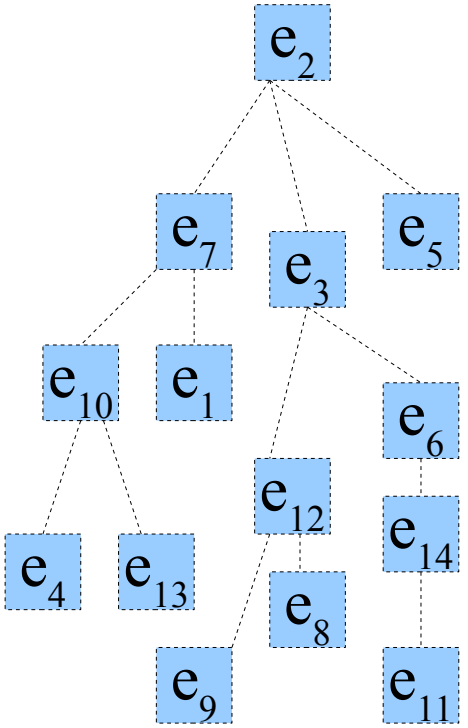
Il modello dei dati

- La base di dati della directory può essere un qualsiasi sistema di archiviazione
  - è definita un'interfaccia (DBI – DataBase Interface)
  - è sufficiente implementarla in un un modulo per pilotare il backend più adatto
- DBI gestisce una serie di **entry** (o **object**)
  - Le entries consistono in una serie di **attribute**
  - Gli attribute implicano un **type** e hanno (molteplici) **value**
  - Ogni entry ha un **DN (Distinguish name)**
- Le entry sono gerarchicamente legate.
  - La posizione è riflessa nel DN
  - La struttura risultante è chiamata **DIT (Dir Information Tree)** e rappresenta il modo con cui viene esposta al mondo la base di dati gestita dalla DBI

DBI e DIT



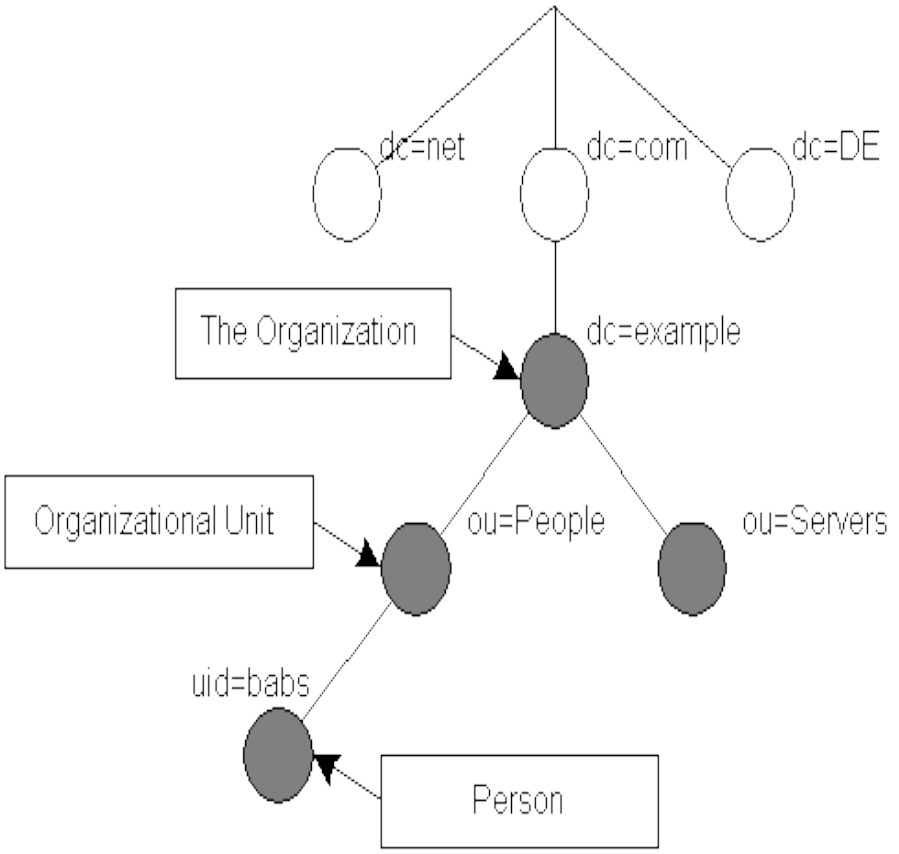
(in linea di principio; normalmente un solo backend è ammesso)



DIT

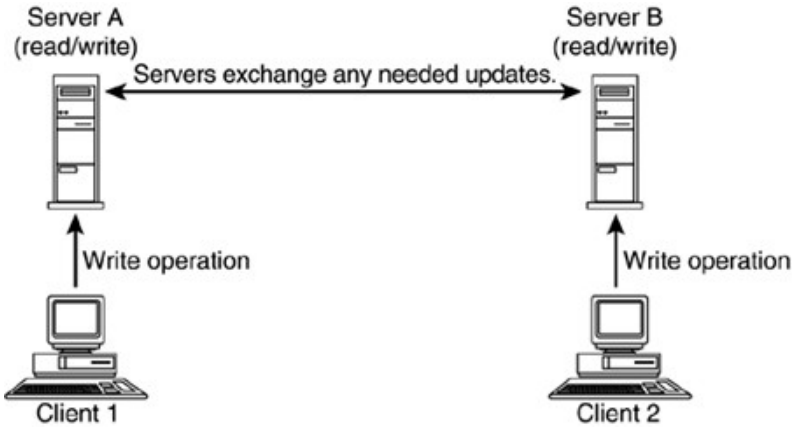
Perché una gerarchia

- Nasce come sistema per organizzare i dati di persone e risorse materiali all'interno di un'organizzazione
- Permette di costruire direttamente riferimenti ai dati
- Permette un facile partizionamento
  - ai fini dell'amministrazione
  - ai fini del controllo degli accessi
  - ai fini della collocazione fisica
- Vincola a una determinata vista dell'organizzazione
  - ma si può scegliere se mitigare il problema adottando un'organizzazione piatta

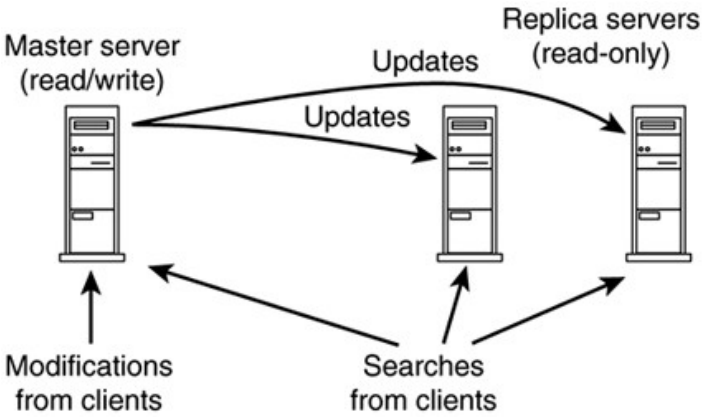


Replica

- Poichè LDAP è un servizio essenziale, prevede nativamente meccanismi per l'elevata disponibilità
- La sua natura (scritture rare) permette di implementare efficacemente la ridondanza senza accorgimenti troppo sofisticati
  - modello corrente: multimaster →
    - tutti i server accettano scritture
    - protocolli ad-hoc per aggiornare reciprocamente i DBI
  - modello storico ma diffuso: master-replica
    - un **master server** accetta scritture
    - molti **replica server** sono utilizzati in sola lettura (è l'operazione che deve scalare ed essere affidabile)
    - semplice copia periodica del DBI master sulle repliche

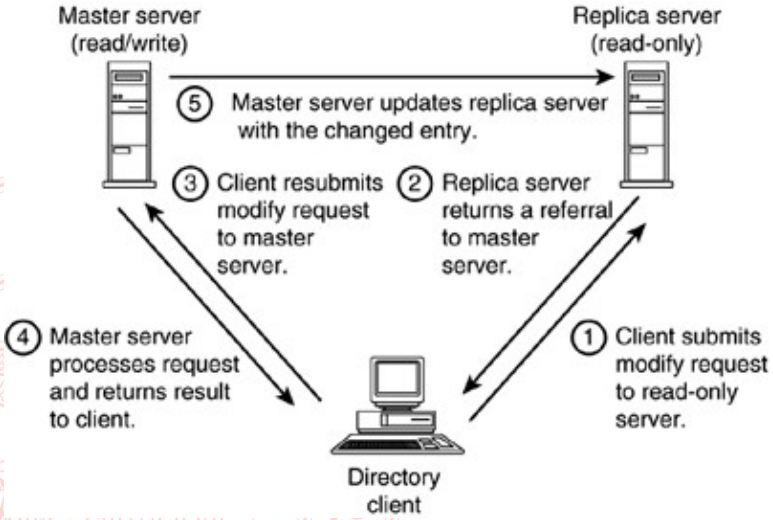


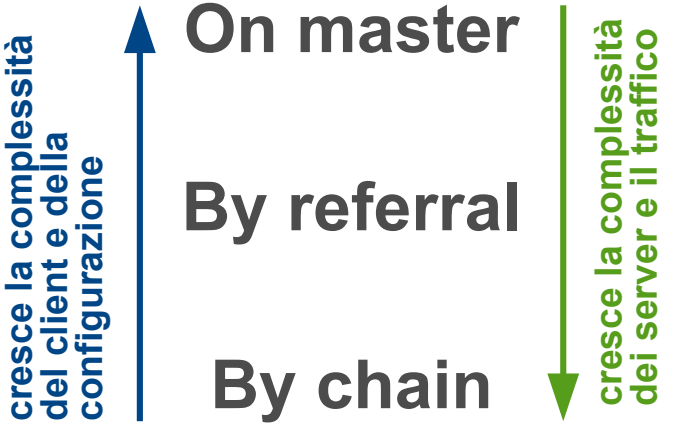
Tre modelli master-replica



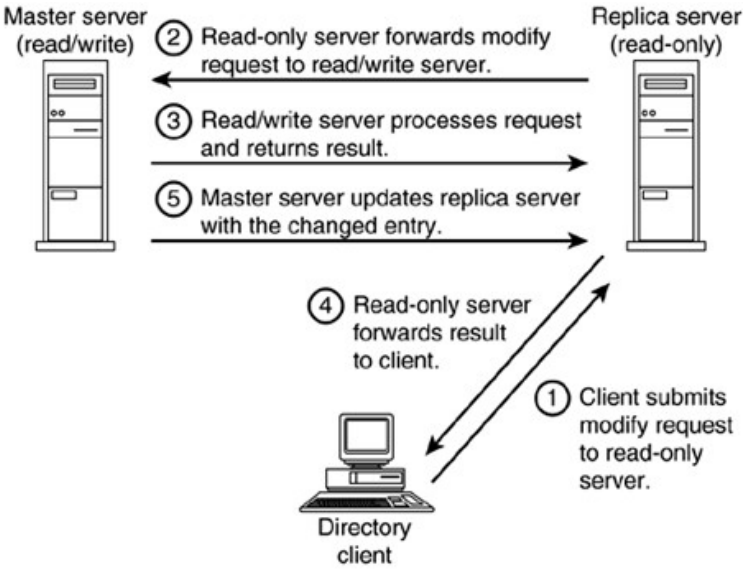
On master

By referral





By chain

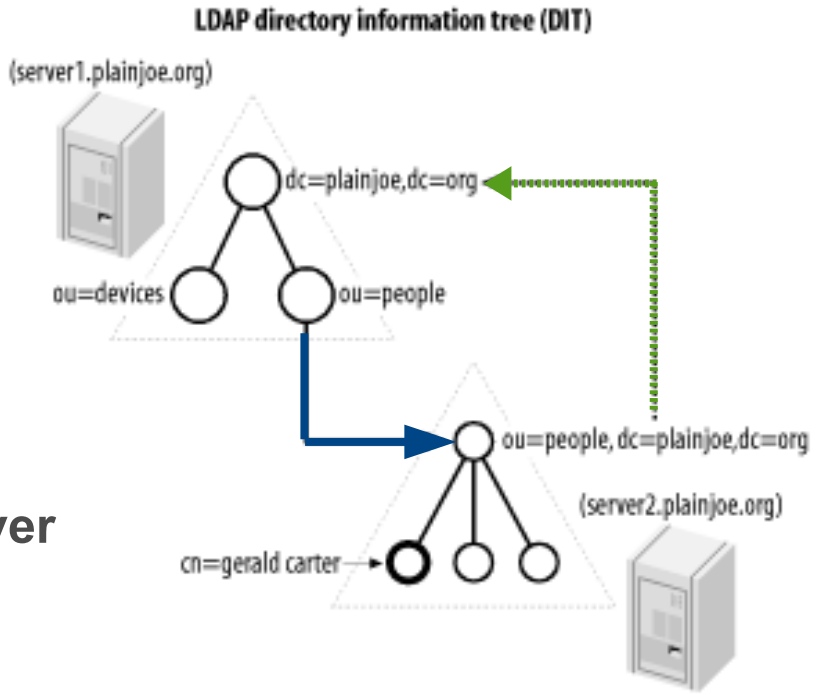


Directory distribuite

- Tre ottimi motivi per sezionare un DIT e collocare diversi sottoalberi su diversi server
  - Prestazioni
  - Località geografica
  - Delega amministrativa
- Sono sufficienti due link:
  - lato server principale

    Subordinate knowledge link
    - Chiamato semplicemente reference, connette logicamente un nodo all'interno di un albero al context naming di un altro server
  - lato server che ottiene la delega

    Superior knowledge link
    - Contiene l'URI del server a cui ci si riferisce per la parte "superiore" della directory



Formato delle entry

- Una entry è una collezione di attributi
  - a differenza di un oggetto di un linguaggio, non si definiscono variabili di un dato tipo, ma si specifica direttamente il valore etichettato dal tipo
- tra gli attributi, ce ne sono due sempre presenti
  - `dn` (distinguished name)
  - `objectClass` (una o più – altri dettagli in seguito)
- esempio:

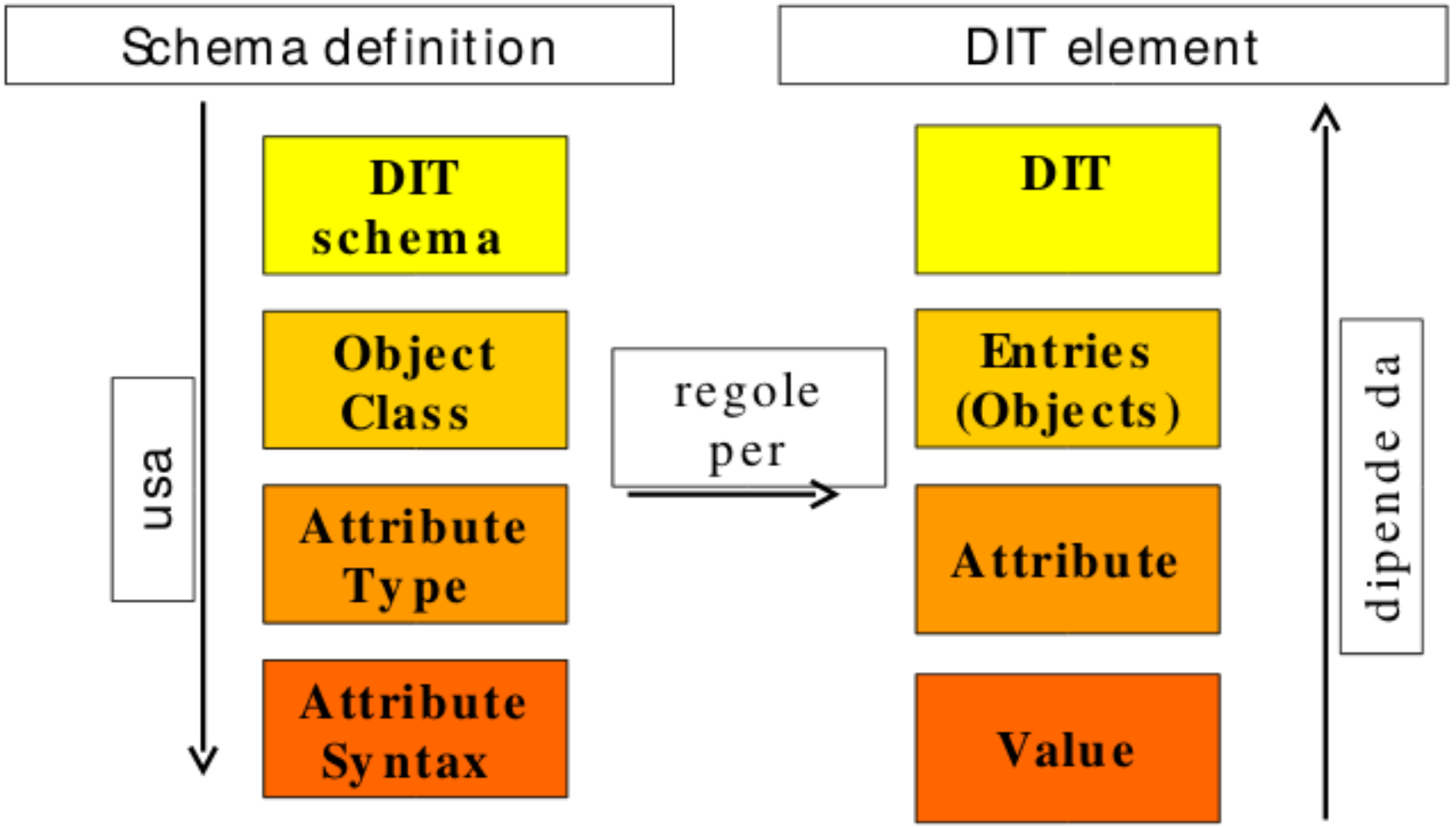
```
dn: dc=labammsis
objectClass: dcObject
objectClass: organization
dc: labammsis
o: universita
```

- questo formato è usato per scambiare entry tra client e server e prende il nome di LDIF (LDAP Interchange Format)
  - un file LDIF può contenere più entry, semplicemente separate da una riga vuota

Schema

- L'inserimento di entry deve essere disciplinato
  - oggetti ben formati
  - visione omogenea e condivisa tra tutti gli utilizzatori
- Si usa uno *schema*
  - insieme di regole che descrivono i dati immagazzinati
  - contiene due tipi di definizioni: `objectClass` e `attributeType`
- Ogni entry è modellata su una o più *objectClass*
  - vincola i tipi di attributi obbligatoriamente o facoltativamente presenti nella entry
- Ogni attributo ha un *attributeType*
  - definisce i tipi di dato e le regole per compararli

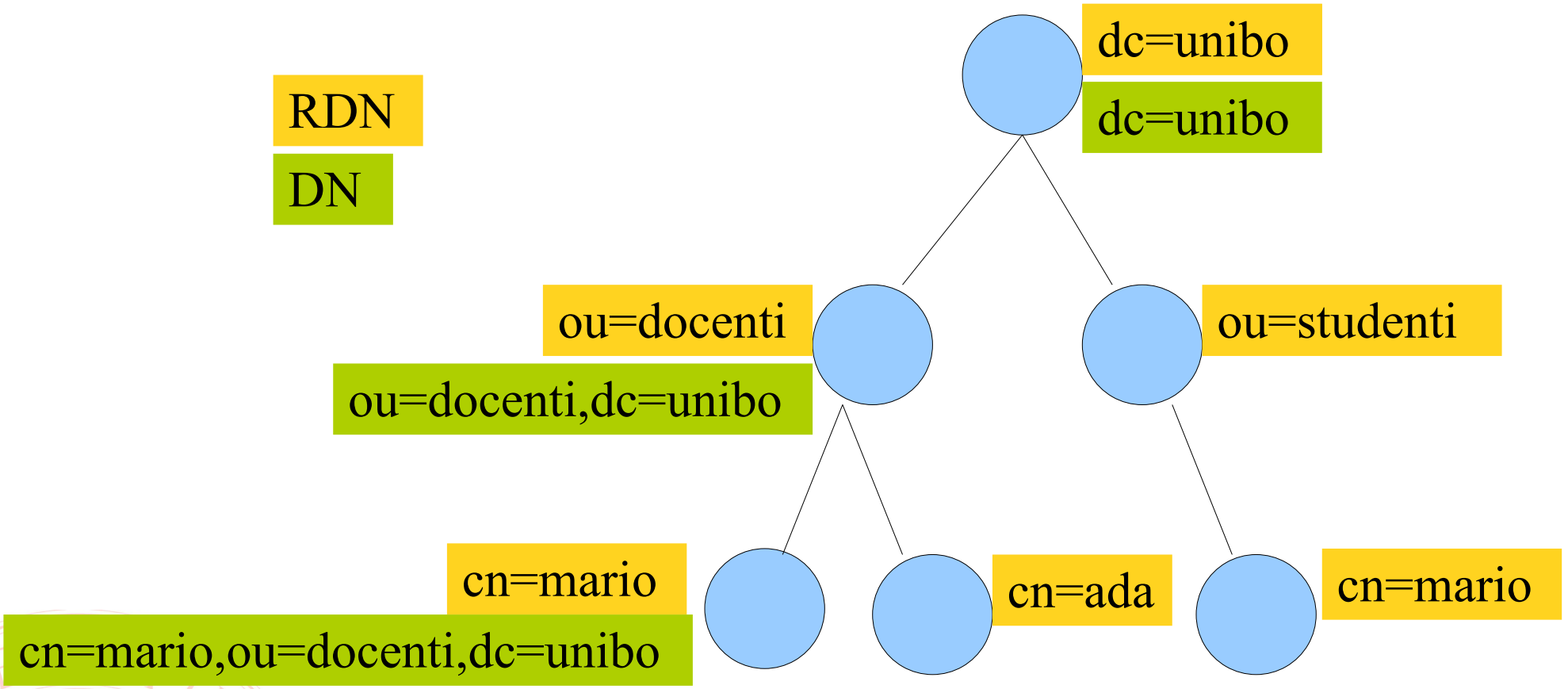
Schema


Schema definition
DIT element
DIT schema
Object Class
Attribute Type
Attribute Syntax
usa
regole per
DIT
Entries (Objects)
Attribute
Value
dipende da

Spazio dei nomi

- In ogni entry si sceglie un attributo rappresentativo
  - sintatticamente: scelta arbitraria
  - in pratica: linea guida per costruire una gerarchia sensata in modo deterministico
- La stringa `tipoAttributo=valoreAttributo` scelta è il *relative distinguished name* della entry (RDN)
- Il nome del nodo dell'albero a cui "appendere" la entry è il *base distinguished name* della entry (BDN)
- Il *distinguished name* è il nome univoco ottenuto concatenando i due DN=RDN+BDN

Spazio dei nomi - esempio



Spazio dei nomi – note

- La radice dell'albero è l'ultimo elemento del DN
  - all'opposto dei nomi assoluti in un filesystem
- Ogni RDN deve essere localmente unico (tra le entry connesse allo stesso BDN) → ogni DN è globalmente unico
- a volte un singolo attributo non è sufficiente a garantire l'unicità del RDN
  - soluzione "da database": introdurre un attributo id univoco
    - contraria alla filosofia "directory"
    - gli attributi rappresentano proprietà reali degli oggetti
  - soluzione sintattica adottata da LDAP: creare il RDN tramite la concatenazione (+) di più coppie attributo=valore
    - es: `cn=Marco+sn=Prandini,ou=docenti,dc=unibo`

Definizione dei tipi di attributo

- Un *attributeType* è simile a un tipo in un linguaggio di programmazione, ma specifica separatamente
  - **SYNTAX**, il vero e proprio tipo di dato nativo

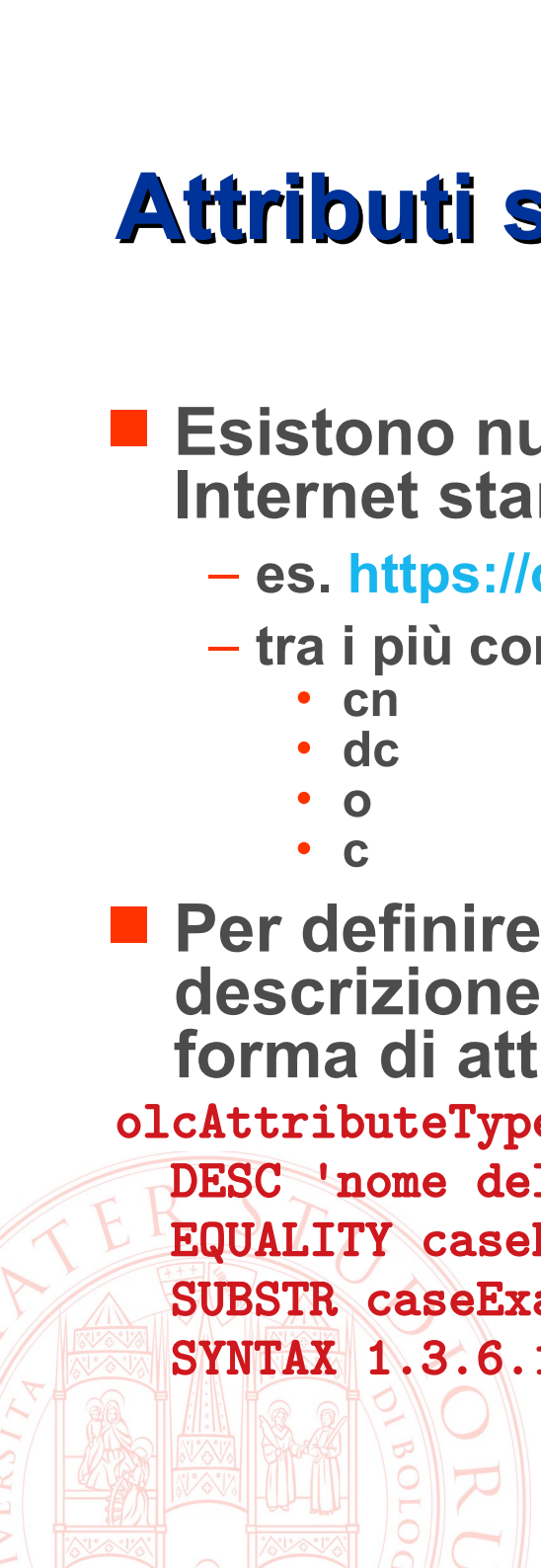
    https://tools.ietf.org/html/rfc4517#section-3
  - *Matching Rules* https://tools.ietf.org/html/rfc4517#section-4,

    le regole per stabilire i criteri di confronto tra valori, necessarie per determinare uguaglianza e ordinamento durante le ricerche
    - **ORDERING**
    - **EQUALITY**
    - **SUBSTRING**
  - Eventuali vincoli d'uso
    - **SINGLE-VALUE** (ammesso un solo valore di questo tipo nella entry)
    - NO-USER-MODIFICATION, USAGE ...
  - Eventuali dipendenze gerarchiche
    - **SUP *<altro attributeType>***
    - il tipo eredita tutte le proprietà del superiore, ne può ridefinire, e nelle ricerche per un tipo superiore vengono restituiti anche tutti i valori dei tipi basati su di esso

Attributi standard e nuove definizioni

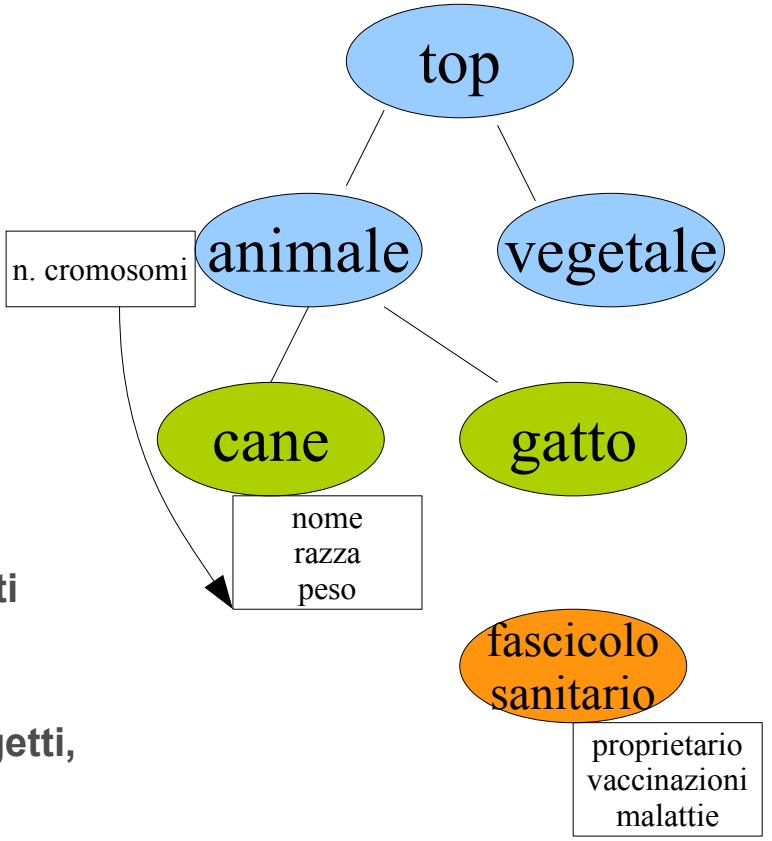
- Esistono numerosissimi attributeType definiti da vari Internet standard per gli usi più comuni
  - es. https://oav.net/mirrors/LDAP-ObjectClasses.html
  - tra i più comuni
    - cn common name
    - dc domain component
    - o organization
    - c country
- Per definire un nuovo attributeType, si inserisce la sua descrizione nella configurazione del server… sotto forma di attributo di una entry speciale!

```
olcAttributeTypes: ( 1000.1.1.1 NAME ( 'fn' 'filename' )
  DESC 'nome del file'
  EQUALITY caseExactMatch
  SUBSTR caseExactSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```



Classi

- Lo scopo essenziale delle objectClass è di elencare
  - quali attributi deve avere una entry: **MUST**
  - quali attributi può avere una entry: **MAY**
- Le classi possono essere di tre tipi
  - **ABSTRACT**
    - servono per creare una tassonomia di categorie di oggetti, ancora troppo astratte per poter originare entry
  - **STRUCTURAL**
    - servono per descrivere categorie di oggetti concreti
  - **AUXILIARY**
    - servono per descrivere collezioni di attributi non direttamente collegati a specifiche categorie di oggetti, ma che possono essere aggiunti per arricchire il contenuto informativo di una entry
- Le classi possono essere definite gerarchicamente
  - **SUP <altra objectClass>**
  - la classe inferiore eredita tutti gli attributi MUST e MAY della superiore e può aggiungerne altri



Classi standard e nuove definizioni

- Esistono numerosissime objectClass definite da vari Internet standard per gli usi più comuni
  - es. https://oav.net/mirrors/LDAP-ObjectClasses.html
  - tra i più comuni
    - organization
    - person
    - device
- Per definire una nuova objectClass, si inserisce la sua descrizione nella configurazione del server… sotto forma di attributo di una entry speciale!

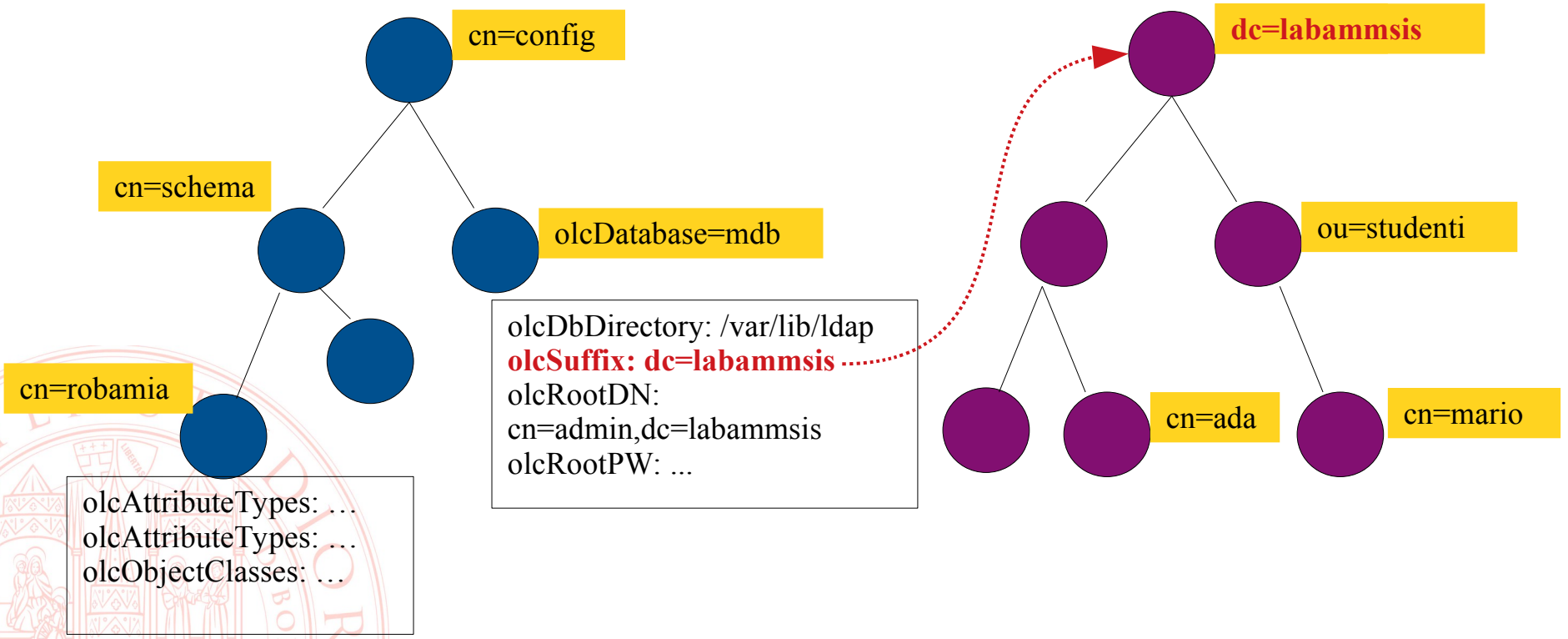
```
olcObjectClasses: ( 1000.2.1.1 NAME 'dir'
  DESC 'una directory'
  MUST fn
  MAY fs
  AUXILIARY )
```

Uso delle classi

- In ogni entry
  - DEVE essere menzionata UNA E UNA sola classe strutturale
  - POSSONO essere menzionate più classi ausiliarie
- È quindi possibile una sorta di ereditarietà multipla
  - ogni objectClass specifica attributeType che devono essere definiti nello stesso schema
  - non ci sono quindi conflitti se più objectClass in una entry menzionano lo stesso attributeType
  - l'effetto è che la entry deve contenere tutti gli attributi MUST di tutte le sue classi, e può contenere tutti gli attributi MAY
  - i vincoli risultanti sono quelli più stringenti (se un attributo è sia nell'insieme MUST che nel MAY, deve essere presente)

Inserimento di nuovi schemi

- LDAP si "autodescrive": quasi tutte le direttive di configurazione sono attributi in entry di un albero separato da quello dei dati veri e propri



Inserimento di nuovi schemi come LDIF

- Per riconfigurare un server LDAP quindi si inserisce una entry in formato LDIF; es. per uno schema si può creare il file `filesystem.ldif`:

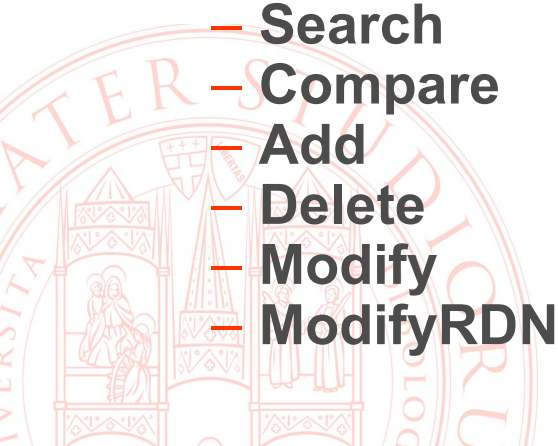
```
dn: cn=filesystem,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: filesystem
olcAttributeTypes: ( 1000.1.1.1 NAME ( 'fn' 'filename' )
  DESC 'nome del file'
  EQUALITY caseExactMatch
  SUBSTR caseExactSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
olcAttributeTypes: ( 1000.1.1.2 NAME ( 'fs' 'filesize' )
  DESC 'dimensioni del file'
  EQUALITY integerMatch
  ORDERING integerOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
olcObjectClasses: ( 1000.2.1.1 NAME 'dir'
  DESC 'una directory'
  MUST fn
  MAY fs
  AUXILIARY )
olcObjectClasses: ( 1000.2.1.2 NAME 'file'
  DESC 'un file'
  MUST ( fn $ fs )
  AUXILIARY )
```

- Solitamente si usa un comando apposito per scavalcare l'autenticazione LDAP e agire da amministratori locali = amministratori del servizio

```
ldapadd -Y EXTERNAL -H ldapi:/// -ffilesystem.ldif
```

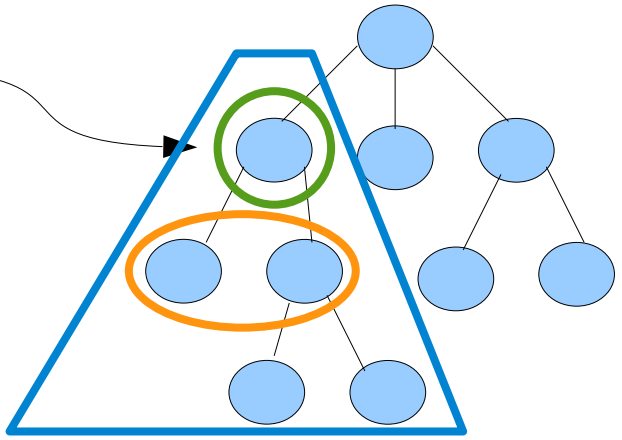
Protocollo LDAP

- Protocollo applicativo su TCP
  - porta 389 (standard)
  - porta 636 (over TLS)
- Basato su sessione
  - bind
    - risposta basata su successo autenticazione o chiusura
  - (serie di)
    - richiesta di operazione
    - risposta (dati)
    - risposta (codice esito)
  - unbind
- Operazioni
  - Search
  - Compare
  - Add
  - Delete
  - Modify
  - ModifyRDN



Search

- La ricerca deve specificare
  - come tutte le operazioni, un **bind DN**
    - equivale all'utente con cui autenticarsi sul server LDAP
  - un **base DN**
    - il punto del DIT da cui iniziare la ricerca
  - uno **scope**
    - quanto estendere la ricerca
    - **sub** – intero sottoalbero, il default
    - **one** – le sole entry figlie dirette del base DN
    - **base** – il solo nodo base DN
  - eventualmente un **filtro**
    - ricerca per contenuto della entry anziché per posizione
    - espressioni logiche in notazione prefissa



Filtri – sintassi ed esempi

```
<filter>::='('<filtercomp>')'
<filtercomp>::=<and>|<or>|<not>|<item>
<and>::='&'<filterlist>
<or>::='|'<filterlist>
<not>::='!'<filter>
<filterlist>::=<filter>|<filter><filterlist>
<item>::=<simple>|<present>|<substring>
<simple>::=<attr><filtertype><value>
<filtertype>::=<equal>|<approx>|<greater>|<less>
<equal>::='='
<approx>::='~='
<greater>::='>='
<less>::='<='
<present>::=<attr>'=*'
<substring>::=<attr>'='<initial><any><final>
<initial>::=NULL|<value>
<any>::='*'<starval>
<starval>::=NULL|<value>'*'<starval>
<final>::=NULL|<value>
```

```
(cn=Babs Jensen)

(!(cn=Tim Howes))

(&
  (objectClass=Person)
  (|
    (sn=Jensen)(cn=Babs J*)
  )
)

(o= univ*of*mich*)

(objectclass=posix Account)
```

```
(&(|(uid=jack)(uid=jill))(objectclass=posix Account)
```

Operazioni e strumenti

- per la ricerca

  `ldapsearch -x -b dc=labammsis [ -s base | one | sub ] [filtro]`

- per l'aggiunta

  `ldapadd -x -D "cn=admin,dc=labammsis" -w admin`

  `[ -f file_ldif_da_inserire ]`

  - se omesso usa stdin

- per la modifica

  `ldapmodify`

  - stessi parametry ldapadd

  – varietà di casi

  - aggiunta/modifica/rimozione attributo

- per la rimozione

  `ldapdelete -x -D "cn=admin,dc=labammsis" -w admin` *DN*